Vol. 37 No. 4　　情　報　処　理　　Apr. 1996

連載講座

算術演算回路のアルゴリズム

4. 初等関数計算回路のアルゴリズム†

—専用回路の実現に向けて—

高　木　直　史††

1. まえがき

第4回は，三角関数や指数・対数関数などの初等関数計算回路のアルゴリズムおよび剰余系演算回路のアルゴリズムについて述べる．

三角関数や指数・対数関数などの初等関数の計算は種々の応用に現われる．計算法としては，多項式近似法（補間）やCORDIC（COordinate Rotation DIgital Computer）法，STL（Sequential Table Look-up）法が用いられている．多項式近似法は関数値を多項式近似によって計算するものであり，計算は積和演算の繰り返しになる．定義域を区間に区切り，各区間ごとに近似多項式を定めるようにすれば，必要な精度を得るための多項式の次数を小さくできる．多項式の係数は，演算数の上位数ビットをインデックスとしてテーブル参照により与えればよい．計算回路は，積和演算器およびテーブル用のROMで構成できる．第2回で述べた中間積の符号変換の手法を用いれば，多項式の評価を高速化できる．CORDIC法およびSTL法は一次の収束法であり，計算はシフトと加減算，定数の読み出しという単純な演算の繰り返しになる．

剰余系での加減算や乗算，べき乗算などは，近年注目されている公開鍵暗号の暗号化，復号処理などで用いられる．公開鍵暗号では，500ビットから1000ビットといった大きな数を法とする剰余系演算が必要である．法が大きい場合は，第1回で述べた桁上げ保存表現や冗長2進表現などの冗長表現の導入による高速化が不可欠である．

初等関数の計算や剰余系演算については，すでに専用回路が種々開発されているが，今後，ますます開発が盛んになるものと考えられる．

本稿では，初等関数計算の専用回路に適した計算法として，次章でCORDIC法，3.でSTL法について述べる．また，4.で剰余加算および剰余乗算回路のアルゴリズムについて述べる．

2. 三角関数計算のためのCORDIC法

CORDIC法はVolderによって三角関数の計算法として提案され[1]，後にWaltherによって双曲線関数などの計算にも拡張した統一アルゴリズムが示された[2]．電卓などで実用されており，専用回路も開発されている．さらに，行列の三角化や特異値分解などへの応用も研究されている[3]．

2.1 正弦および余弦の計算

まず，三角関数の計算について考える．**図-1**に示すように，X-Y平面上に原点Oと二点P（(X, Y)）およびP'（(X', Y')）からなる直角三角形OPP'があり，$\angle POP' = \arctan 2^{-k}$であるとき，$X' = X - Y \cdot 2^{-k}$，$Y' = Y + X \cdot 2^{-k}$となり，ベクトル$\overrightarrow{OP'}$は，ベクトル$\overrightarrow{OP}$を原点$O$を中心として正方向（反時計回り）に角度$\arctan 2^{-k}$回転し，$(1+2^{-2k})^{1/2}$倍に拡張したものになっている．CORDIC法による三角関数の計算においては，この「回転・拡張」が基本操作となる．

正弦および余弦を計算する場合は，**図-2**に示すように，X軸上の点から始めて，この点に対して原点を中心とする角度$\arctan 2^{-j}$（$j=0, 1, 2, \cdots$）の正または負方向の回転・拡張を施す．これをCORDIC法の回転モードと呼ぶ．$\sin\theta$および$\cos\theta$（$|\theta| \leq \frac{\pi}{2}$）を計算する場合，回転角の合計が$\theta$，最終点の原点からの距離が1になるようにすると，最終点のY座標およびX座標が求め

† Algorithms for Arithmetic Circuits 4 Algorithms for Elementary Function Generators by Naofumi TAKAGI (Department of Infomation Engineering, Graduate School of Engineering, Nagoya University).

†† 名古屋大学大学院工学研究科情報工学専攻

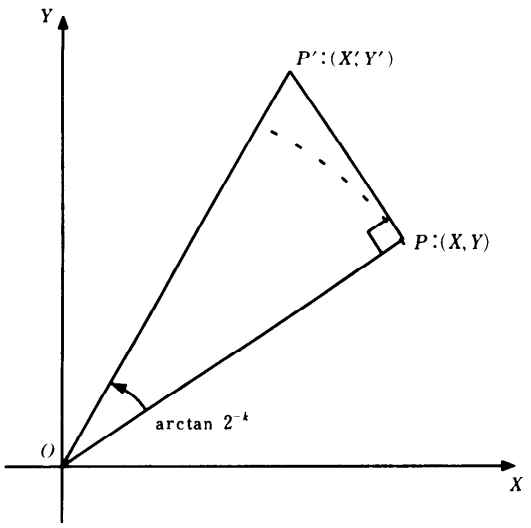

図-1 CORDIC 法における回転・拡張

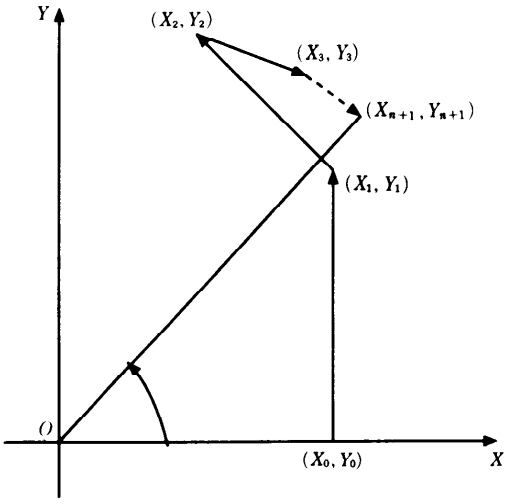

図-2 CORDIC 法による正弦および余弦の計算

る値となる．

点の座標を (X_j, Y_j) とし，回転の残り角を Z_j とする．初期値は，$Y_0: =0$，$Z_0: =\theta$ である．X_0 については後述する．各回転・拡張による点の移動および回転の残り角は，次の漸化式で表される．

$$X_{j+1}: =X_j-q_j\cdot 2^{-j}\cdot Y_j$$
$$Y_{j+1}: =Y_j+q_j\cdot 2^{-j}\cdot X_j$$
$$Z_{j+1}: =Z_j-q_j\cdot\arctan 2^{-j}$$

ここに，q_j は j 回目の回転の方向であり，$q_j=1$ なら正方向，$q_j=-1$ なら負方向の回転を示す．$\arctan 2^{-j}$ $(j=0, 1, 2, \cdots)$ は定数であるから，あらかじめ計算し，記憶しておく．漸化式の計算は，シフトと加減算，定数の読み出しによって行うことができる．回転の方向 q_j は，Z_{j+1} が 0 に近付くように，$\{-1, 1\}$ の中から選ぶ．すなわち，Z_j が非負ならば q_j を 1，負ならば -1 とする．各 j について，必ず正または負の回転・拡張を行う．j 回目の回転・拡張により，回転の方向に関係なく，ベクトルの長さは $(1+2^{-2j})^{1/2}$ 倍される．$2^{-j-1}<\arctan 2^{-j}<2\cdot\arctan 2^{-j-1}<2^{-j}$ であり，$|Z_j|<2\cdot\arctan 2^{-j}<2^{-j+1}$ となり，一次の収束が保証される．

小数点以下 n ビットの精度を得るためには，およそ $n+1$ 回の回転を行う必要がある．$n+1$ 回の回転・拡張でベクトルの長さは $K=\prod_{j=0}^{n}(1+2^{-2j})^{1/2}$ 倍になる．K は θ の値によらず一定である．そこで，X_j の初期値として $X_0: =1/K$ として最初のベクトルの長さを $1/K$ としておけば，最終的なベクトルの長さが 1 となる．

1 ステップ分の計算を行う回路は，X_j，Y_j，Z_j を保持するための 3 つのレジスタ，漸化式の計算のための 2 つのシフタと 3 つの加減算器，$\arctan 2^{-j}$ を記憶する ROM によって構成できる．($j>n/3$ に対しては，$\arctan 2^{-j}=2^{-j}$ としてよいので，記憶する定数はおよそ $n/3$ 個でよい．) 1 ステップ分の回路の段数は，加減算器として順次桁上げ加算器を用いると n に比例し，桁上げ先見加算器を用いると $\log n$ に比例する．ただし，回路の段数が $\log n$ に比例するようにするには，シフタとして高速のバレルシフタ（1 度に多数桁のシフトが可能なシフタ）を用いる必要がある．数ステップ分の計算を行う回路を構成することも考えられる．

X_0 を適当において，ベクトル $\overrightarrow{OP_j}$ の長さが 1 になるようにすると，$\cos\theta=X_j\cdot\cos Z_j-Y_j\cdot\sin Z_j$ および $\sin\theta=Y_j\cdot\cos Z_j+X_j\cdot\sin Z_j$ が成り立つ．$|Z_j|<2^{-j+1}$ であるから，$j>n/2$ である j に対して，$\cos Z_j(=1-Z_j^2/2+\cdots)\approx 1$，$\sin Z_j(=Z_j-Z_j^3/3+\cdots)\approx Z_j$ となり，$\cos\theta=X_j-Y_j\cdot Z_j$，$\sin\theta=Y_j+X_j\cdot Z_j$ となる．したがって，$j>n/2$ に対する計算を 2 回の乗算と加算，減算に置き換えることができる[4),5)]．

上記では X 軸上の点から始めたが，θ の上位数ビットからテーブル参照により初期近似座標を求め，この点から CORDIC 法を適用することに

より高速化することも考えられる．

2.2 冗長表現を用いた高速化

第1回で述べ，前回までにもしばしば現れた，桁上げ保存形や冗長2進表現などの冗長表現を用いることにより，CORDIC法による計算を高速化できる．すなわち，X_j，Y_jおよびZ_jを桁上げ保存形または冗長2進表現で表し，漸化式の計算における加減算を桁上げの伝搬なしに高速に行う．これは，冗長CORDIC法[3]と呼ばれている．

桁上げ保存形や冗長2進表現では，正確な正負判定のためには最悪の場合全桁を調べる必要があり，大きな計算時間を要する．したがって，計算を高速化するためには，回転の方向q_jの決定においてZ_jの正確な正負判定が不要となるように，CORDIC法を改造する必要がある．第3回で述べたSRT除算法の場合と同様に，q_jを$\{-1, 0, 1\}$の中から選択するようにすれば，Z_jの上位数桁の値からq_jを決定することができる．しかし，この場合，$q_j=\pm 1$のときは回転・拡張によりベクトルの長さが$(1+2^{-2j})^{1/2}$倍に拡張されるのに対し，$q_j=0$のときは回転・拡張を行わないのでベクトルの長さはそのままである．したがって，ベクトルの長さの倍率が演算数θによって一定にならず，倍率の計算とこれによる計算値の補正が必要となる．これらの計算には開平や除算が必要であり，大きな計算時間がかかる．

冗長CORDIC法において，ベクトルの長さの倍率が演算数によらず一定になるようにした計算法として，二重回転法と補正回転法が提案されている[6]．

二重回転法では，回転の方向q_jを$\{-1, 0, 1\}$から選択する．j回目の回転を$\arctan 2^{-j}$の1回の回転・拡張の代わりに$\arctan 2^{-j-1}$の2回の回転・拡張で行う．すなわち，$q_j=1$のとき$\arctan 2^{-j-1}$の正方向への2回の回転・拡張，$q_j=-1$のとき負方向への2回の回転・拡張，$q_j=0$のとき正方向と負方向への1回ずつの回転・拡張を行う．これにより，回転の方向によらずベクトルの長さが必ず$1+2^{-2j-2}$倍に拡張される．したがって，ベクトルの長さの倍率が演算数によらず一定になる．冗長2進表現を用いた場合，Z_jの上位3桁の値からq_jを決定できる．

補正回転法では，通常のCORDIC法と同様，q_jを$\{-1, 1\}$から選択し，各回転は$\arctan 2^{-j}$の1回の回転・拡張により行う．Z_jが上位数桁からでは正負判定ができない場合は，強制的に$q_j:=1$とする．本来$q_j:=-1$として負方向に回転すべきところを$q_j:=1$として正方向に回転してしまうと，収束が保証されなくなる．そこで，何回かに1度，補正のための回転を行う．k回に1度補正回転を行う場合，k個に1つのjについて，$\arctan 2^{-j}$の回転・拡張を2回行うことになる．冗長2進表現を用いた場合，q_jの決定に，Z_jの上位，最大$k+2$桁を調べる必要がある．

2.3 逆正接の計算

$X-Y$平面上の点(X_0, Y_0)から始めて，$\arctan 2^{-j}$ $(j=0, 1, 2, \cdots)$の回転・拡張を行い，Y_jを0に近付けていく（点をX軸に近付けていく）と，Z_jは$Z_0+\arctan(Y_0/X_0)$に近付いていく．これをCORDIC法のベクトルモードと呼ぶ．ここで，初期値として$X_0:=1$，$Y_0:=Y$，$Z_0:=0$とすると，$\arctan Y$が求まる．小数点以下nビットの精度を得るためには，およそ$n+1$回の回転を行う必要がある．

漸化式は正弦および余弦の計算の場合とまったく同じである．回転の方向q_jは，Y_{j+1}が0に近付くように，$\{-1, 1\}$の中から選ぶ．すなわち，Y_jが非負ならばq_jを-1，負ならば1とする．漸化式の計算は，シフトと加減算，定数の読み出しによって行うことができる．

逆正接の計算にも，桁上げ保存形や冗長2進表現などの冗長表現を導入して計算を高速化できる．逆正接の計算では，ベクトルの長さは計算値に影響しないので，回転・拡張の回数を一定にする必要はない．したがって，回転の方向q_jをY_jの上位数桁より$\{-1, 0, 1\}$の中から定め，0の場合は回転・拡張を行わなくてよい．

2.4 統一アルゴリズム

三角関数の計算においては，平面上の円に沿った点の移動を考えた．これを拡張し，平面上の直線および双曲線に沿った点の移動をも考え，これらを統一した，統一CORDIC法が提案されている[2]．点の移動はパラメータmを用いて，以下の漸化式で表される．

$$X_{j+1}:=X_j-m\cdot q_j\cdot 2^{-j}\cdot Y_j$$
$$Y_{j+1}:=Y_j+q_j\cdot 2^{-j}\cdot X_j$$

$Z_{j+1} := Z_j - q_j \cdot (1/\sqrt{m})\arctan(\sqrt{m}\cdot 2^{-j})$

$m=1$ のときは円に沿った移動で，回転モードにより $K_1(X_0\cos Z_0 - Y_0\sin Z_0)$ と $K_1(X_0\sin Z_0 + Y_0\cos Z_0)$，ベクトルモードにより $K_1\sqrt{X_0^2+Y_0^2}$ と $Z_0+\arctan(Y_0/X_0)$ を計算できる．したがって，上述のように，正弦および余弦，逆正接などを計算できる．

$m=0$ のときは直線に沿った移動で，回転モードにより $Y_0+X_0\cdot Z_0$，ベクトルモードにより Z_0+Y_0/X_0 を計算できる．したがって，乗算（積和演算）と除算を実現できる．

$m=-1$ のときは双曲線に沿った移動で，回転モードにより $K_{-1}(X_0\cosh Z_0+Y_0\sinh Z_0)$ と $K_{-1}(X_0\sinh Z_0+Y_0\cosh Z_0)$，ベクトルモードにより $K_{-1}\sqrt{X_0^2-Y_0^2}$ と $Z_0+\tanh^{-1}(Y_0/X_0)$ を計算できる．したがって，双曲線正弦および双曲線余弦，双曲線逆正接などを計算できる．

3. 指数・対数関数計算のためのSTL法

STL (Sequential Table Look-up) 法は，逐次表を引きながら指数関数および対数関数の計算を行うもので，Cantorらによって提案され，後にSpeckerやChenによって改良，拡張された[4),7)]．

任意の数 P, Q (>0), A (>0) に対して，$P+\ln Q=(P-\ln A)+\ln(Q\cdot A)$ が成り立つ．($\ln$ は自然対数を表す．) STL法では，$P_0+\ln Q_0=P_1+\ln Q_1=\cdots=P_n+\ln Q_n$ という式の変換を考える．この変換は，次の漸化式で表される．

$P_j := P_{j-1}-\ln A_j$

$Q_j := Q_{j-1}\cdot A_j$

A_j を $\{1, 1+2^{-j}\}$ あるいは $\{1, 1-2^{-j}\}$ から選び，定数 $\ln(1+2^{-j})$ あるいは $\ln(1-2^{-j})$ をあらかじめ計算しテーブルに記憶しておけば，漸化式の計算がシフトと加減算，定数の読み出しにより行える．

$X(0\le X<\ln 2)$ に対して $\exp X$ を計算する場合，初期値を $P_0 := X$, $Q_0 := 1$ とし，各ステップでは，P_j が0に近付くように A_j を選ぶ．すなわち，各ステップでは，$P_{j-1}-\ln(1+2^{-j})\ge 0$ ならば $A_j := 1+2^{-j}$, <0 ならば $A_j := 1$ とする．$2^{-j-1}<\ln(1+2^{-j})<2\cdot\ln(1+2^{-j-1})<2^{-j}$ であり，$0\le P_j<2^{-j}$ となり，一次の収束が保証される．小数点以下 n ビットの精度を得るには，このステップをおよそ n 回繰り返せばよい．このとき，$P_n\approx 0$ となり，$X+\ln 1\approx 0+\ln Q_n$，すなわち，$X\approx\ln Q_n$ が成り立ち，$\exp X\approx Q_n$ となる．

$X(1\le X<2)$ に対して $\ln X$ を計算する場合，初期値を $P_0 := 0$, $Q_0 := X$ とし，各ステップでは，Q_j が1に近付くように A_j を選ぶ．すなわち，各ステップでは，$Q_{j-1}-2^{-j}\cdot Q_{j-1}\ge 1$ ならば $A_j := 1-2^{-j}$, <1 ならば $A_j := 1$ とする．$0\le Q_j-1<2^{-j}$ となり，一次の収束が保証される．小数点以下 n ビットの精度を得るには，このステップをおよそ n 回繰り返せばよい．このとき，$Q_n\approx 1$ となり，$0+\ln X\approx P_n+\ln 1$，すなわち，$\ln X\approx P_n$ となる．

上記では，$\exp X$ の計算においては A_j を $\{1, 1+2^{-j}\}$ から選び，$\ln X$ の計算においては $\{1, 1-2^{-j}\}$ から選んだ．これは，減算シフト型除算法における回復型除算法に相当する．A_j を $\{1-2^{-j}, 1+2^{-j}\}$ から選ぶ手法や $\{1-2^{-j}, 1, 1+2^{-j}\}$ から選ぶ手法も考えられる．

1ステップ分の計算を行う回路は，X_j, Y_j を保持するための2つのレジスタ，漸化式の計算のための1つのシフタと2つの加減算器，$\ln(1+2^{-j})$ あるいは $\ln(1-2^{-j})$ を記憶するROMによって構成できる．数ステップ分の計算を行う回路を構成することも考えられる．

$\exp X$ の計算においては，$X=P_j+\ln Q_j$，すなわち，$\exp X=Q_j\cdot\exp P_j$ が成り立つ．$|P_j|<2^{-j}$ であるから，$j>n/2$ である j に対して，$\exp P_j(=1+P_j+P_j^2/2+\cdots)\approx 1+P_j$ となり，$\exp X=Q_j\cdot(1+P_j)$ となる．したがって，$j>n/2$ に対する計算を1回の乗算に置き換えることができる．また，$\ln X$ の計算においては，$\ln X=P_j+\ln Q_j$ が成り立つ．$|Q_j-1|<2^{-j}$ であるから，$j>n/2$ である j に対して，$\ln Q_j(=(Q_j-1)-(Q_j-1)^2/2+\cdots)\approx Q_j-1$ となり，$\ln X=P_j+Q_j-1$ となる．したがって，$j>n/2$ に対する計算を1回の加算に置き換えることができる[4),5)]．

STL法も桁上げ保存形や冗長2進表現などの冗長表現を用いることにより計算を高速化することができる．この場合，A_j は $\{1-2^{-j}, 1, 1+2^{-j}\}$ から選ぶ．$\exp X$ の計算においては，$P_{j-1}(|P_{j-1}|<2^{-j+1})$ の上位数桁の値から A_j を決定する．$\ln X$ の計算においては，$Q_{j-1}-1(|Q_{j-1}-1|<2^{-j+1})$ の上位数桁の値から A_j を決定する．

4. 剰余系演算回路のアルゴリズム

本章では，最も基本的な剰余系演算である剰余加算と剰余乗算のアルゴリズムについて述べる．法 M は n ビットで，$2^{n-1} \leq M < 2^n$ であるとする．

4.1 剰余加算回路のアルゴリズム

剰余加算は，X，$Y(\in Z_M = \{0, 1, \cdots, M-1\})$ に対し，$X+Y \bmod M$ を計算する演算である．

最も単純な方法は，まず通常の加算を行い，和が M 以上であれば M を減ずるものである．1ないし2回の通常の加算と1回の比較が必要である．

剰余加算を繰り返す場合は，各剰余加算においては，まず通常の加算を行い，最上位から桁上げが生じれば M を減ずればよい．ただし，最終結果が M 以上になれば，M を減じる．各剰余加算における比較が桁上げの検出に置き換えられている．

上記の計算法では，法が大きい場合，通常の加算における桁上げ伝搬のため多大の計算時間を要する．剰余加算を繰り返す場合，法が大きいときには，桁上げ保存形や冗長2進表現などの冗長表現の導入が有効である．冗長表現を用いる場合，Z_M の各要素を冗長符号化する．冗長符号化は種々考えられるが，ここでは冗長2進表現を用いた冗長符号化の一例と，この符号化を用いた剰余加算アルゴリズムを示す．

0を除く Z_M の各要素 A を，値が A または $A-M$ であるような $n+1$ 桁の冗長2進数で表す．(0は，[00…0]で表す．) 値 A および $A-M$ のそれぞれに対して，いくつかの冗長2進表現が存在する．この冗長符号化の下で，被加数 X と加数 Y から和 S を求める剰余加算について考える．アルゴリズムは，2つのステップからなる．第一ステップでは，X と Y を冗長2進数体系で桁上げの伝搬なしに加算し，$n+2$ 桁の冗長2進数 T を得る．第二ステップでは，T の上位3桁の値 tv の負，零，正に応じて，T に M，0，または，$-M$ を冗長2進数体系で桁上げの伝搬なしに加え，S を得る．M は通常の2進数であるから，この加算は通常の冗長2進加算より容易である．S を $n+1$ 桁にするために，最上位桁では特別な計算規則を用いる．$tv<0$ のとき $-2M < T < 0$，$tv=0$ のとき $-2^{n-1} < T < 2^{n-1}$，$tv>0$ のとき $0 < T < 2M$ であるから，$-M < S < M$ となる．また，$S \equiv T \equiv X+Y \pmod M$ である．

4.2 剰余乗算回路のアルゴリズム

剰余乗算は，X，$Y(\in Z_M)$ に対し，$X \cdot Y \bmod M$ を計算する演算である．

最も単純な方法は，まず通常の n ビット2進乗算を行い，次にその積の剰余を求めるものである．剰余の計算は，除算によって行うか，もしくは，あらかじめ $2^i \bmod M (n \leq i \leq 2n-1)$ を計算しておき，その読み出しと剰余加算によって行うことが考えられる．

上記の手法では，中間結果として $2n$ ビットの数を保持する必要があり，法が大きい場合には大きなレジスタが必要になる．そこで，乗算の加算ステップと剰余計算の減算ステップをインタリーブし，乗数 $Y(=[y_{n-1}y_{n-2}\cdots y_0])$ の上位ビットから順に逐次計算を行うことを考える．乗数の1ビットずつ計算を行う場合は，累算結果の2倍に部分積を加えて剰余を計算するステップを n 回繰り返す．すなわち，P_n: $=0$ とし，j: $-n-1, n-2, \cdots, 0$ について，P_j: $=(2 \cdot P_{j+1} + y_j \cdot X) \bmod M$ という計算を行う．

計算の高速化手法として，以下のような手法が知られている[8]．(1)剰余計算において，法 M の何倍を減ずるかを，和と M の上位数ビットのみから決定する．(2)累算結果 P_j を桁上げ保存形や冗長2進表現などの冗長表現で表し，加算を桁上げの伝搬なしに行う．(3)1度に乗数の数ビット分の計算を行う．(4) M の代わりに $K \cdot M$ (K は2のべき乗) を法とし，P_j の冗長度を増して，和に対する部分積 $y_j \cdot X$ の影響を小さくすることにより，剰余計算を容易にする．

法が大きい場合は，冗長表現の導入が不可欠である．1度に数ビット分の計算を行うと，ステップ数は少なくなるが，各ステップでの計算は複雑になる．上述の手法を用い，2ビット分ずつ計算を行い，部分積の生成および法の倍数の生成をシフトと正負反転のみで行うことを可能とした剰余乗算アルゴリズムが提案されている[9]．

剰余乗算の専用回路としては，上記の1ステップの計算を1クロックで行うものが考えられる．回路は，レジスタや冗長2進加算器または桁上げ保存加算器などで構成でき，集積回路化に適した

規則正しいビットスライス構造になる．数ステップ分の計算を1クロックで行う回路を構成することも考えられる．他の構成法として，たとえば1ステップで32ビット分の計算を行うようにし，32×32ビットの乗算器を用いて1ステップ分の計算を数十クロックで行うものも考えられる．

上記とは異なる剰余乗算法としてMontgomeryのアルゴリズムが提案されており[10]，これに基づく剰余乗算器の構成法も研究されている[8]．

Bを，法Mと互いに素で，Mより大きく，剰余計算が容易な整数とする．Mがnビットで奇数ならば，$B=2^n$とすればよい．Mを法とするBの乗法に関する逆数をB^{-1}とする．すなわち，$B\cdot B^{-1} \bmod M=1$かつ$0<B^{-1}<M$とする．また，M'を$0<M'<B$かつ$B\cdot B^{-1}-M\cdot M'=1$を満たす整数とする．このとき，$0\leq T<M\cdot B$である整数$T$に対し，$T\cdot B^{-1} \bmod M$は，以下の計算により容易に求められる．

function $REDC(T)$

$Q:=(T \bmod B)\cdot M' \bmod B$

$R:=(T+Q\cdot M)/B$

if $R\geq M$ then return $R-M$ else return R□

$B=2^n$であれば，mod Bの計算は単に下位nビットを取ればよい．$Q\cdot M\equiv T\cdot M'\cdot M\equiv -T(\bmod B)$であるから，$T+Q\cdot M$は$B$で割り切れ，$(T+Q\cdot M)/B$は整数になる．$REDC(T)=T\cdot B^{-1} \bmod M$となる．

整数X^*およびY^* $(0\leq X^*, Y^*<M)$が与えられとき，$Z^*:=REDC(X^*\cdot Y^*)$を計算すると，$Z^*=(X^*\cdot Y^*)\cdot B^{-1} \bmod M$となる．すなわち，$(X^*\cdot B^{-1})\cdot(Y^*\cdot B^{-1})\equiv Z^*\cdot B^{-1}(\bmod M)$となる．$(0\leq Z^*<M$である．$)$ここで，$X^*=X\cdot B \bmod M(0\leq X<M)$であるとすると，$X$と$X^*$は1対1対応し，$X^*$は$X$の1つの表現になっている．したがって，$Z^*$はこの表現系での$X$と$Y$の積になっている．

XからX^*への変換は，$X^*:=REDC(X\cdot(B^2 \bmod M))$という計算によって行える．また，$Z^*$から$Z$への変換は，$Z:=REDC(Z^*)$という計算によって行える．公開鍵暗号でしばしば用いられる剰余べき乗算のように，剰余乗算を繰り返し行う場合は，これらの変換は計算全体の最初と最後に行うだけでよい．$B^2 \bmod M$はあらかじめ計算しておけばよい．

Mがnビットの奇数で$B=2^n$であるとき，$REDC(X^*\cdot Y^*)$は以下のように乗数Y^*の下位ビットから順に逐次計算により求めることができる．Y^*は2進表現で$[y_{n-1}y_{n-2}\cdots y_0]$であるとする．乗数の1ビットずつ計算を行う場合は，累算結果に部分積を加え，和が偶数ならこれを2で割り，奇数ならこれに法Mを加えてから2で割るというステップをn回繰り返す．すなわち，$P_0:=0$とし，$j:=0,1,\cdots,n-1$について，P_{j+1}が整数となるように，$P_{j+1}:=(P_j+y_j\cdot X^*)/2$または$P_{j+1}:=(P_j+y_j\cdot X^*+M)/2$とする．$2^j\cdot P_j\equiv[y_{j-1}y_{j-2}\cdots y_0]\cdot X^*(\bmod M)$および$0\leq P_j<M+X^*<2M$が成り立つ．累算結果$P_j$を桁上げ保存形や冗長2進表現などの冗長表現で表し，加算を桁上げの伝搬なしに行うことにより，計算を高速化できる．1度に乗数の数ビット分の計算を行うことも考えられる．

大きな数を法とする剰余乗算は，公開鍵暗号の暗号化・復号処理における基本演算であり，高速計算の要求が高まっている．上記では専用回路のためのアルゴリズムについて述べたが，マイクロプロセッサなどでソフトウェアにより剰余乗算を高速に行う手法の開発も重要な研究課題となっている．大きな数を法とする剰余乗算は，多倍長乗算と除算によって行え，多倍長除算の高速化が剰余乗算の高速化の鍵となる．高速化の手法として，計算の中間結果を0からMの範囲ではなく，$-M$からM，あるいは，0から$2M$の範囲で表すなどの工夫が考えられる[11]．多倍長乗算における総和計算と剰余計算のための多倍長除算の演算順序を工夫した，高速で，かつ，計算に必要なメモリ量の少ないアルゴリズムも開発されている[11]．また，上述のMontgomeryのアルゴリズムでは，多倍長除算を不要にし，計算を高速化している[10]．

5. むすび

連載講座の第4回として，初等関数計算回路および剰余系演算回路のアルゴリズムについて述べた．

今後，初等関数計算や剰余系演算の専用回路の開発が盛んになるものと考えられる．本稿が専用回路を設計する際に参考になれば幸いである．

参 考 文 献

1) Volder, J.: The CORDIC Trigonometric Computing Technique, IRE Trans. Electronic Computers, Vol. EC-8, No. 3, pp. 330-334 (Sep. 1959).
2) Walther, J. S.: A Unified Algorithm for Elementary Functions, Proc. AFIPS Spring Joint Computer Conf., pp. 379-385 (1971).
3) Ercegovac, M. D. and Lang, T.: Redundant and On-line CORDIC: Application to Matrix Triangulation and SVD, IEEE Trans. Computers, Vol. 39, No. 6, pp. 725-740 (June 1990).
4) Chen, T. C.: Automatic Computation of Exponentials, Logarithms, Ratios and Square Roots, IBM J. Research and Development, Vol. 16, No. 4, pp. 380-388 (July 1972).
5) 一松信: 初等関数の数値計算, 教育出版 (1974).
6) Takagi, N., Asada, T. and Yajima, S.: Redundant CORDIC Methods with a Constant Scale Factor for Sine and Cosine Computation, IEEE Trans. Computers, Vol. 40, No. 9, pp. 989 995 (Sep. 1991).
7) Specker, W. H.: A Class of Algorithms for ln x, exp x, sin x, cos x, $\tan^{-1}$ x and $\cot^{-1}$ x, IEEE Trans. Electronic Computers, Vol. EC-14, No. 1, pp. 85-86 (Jan. 1965).
8) Eldridge, S. E. and Walter, C. D.: Hardware Implementation of Montgomery's Modular Multiplication Algorithm, IEEE Trans. Computers, Vol. 42, No. 6, pp. 693-699 (June 1993).
9) Takagi, N.: A Radix-4 Modular Multiplication Hardware Algorithm for Modular Exponentiation, IEEE Trans. Computers, Vol. 41, No. 8, pp. 949-956 (Aug. 1992)
10) Montgomery, P. L.: Modular Multiplication without Trial Division, Mathematics of Computation, Vol. 44, No. 170, pp. 519-521 (Apr. 1985).
11) Takagi, N.: A Modular Multiplication Algorithm with Triangle Additions, Proc. 11 th IEEE Symposium on Computer Arithmetic, pp. 272-276 (June 1993).

(平成7年8月28日受付)

高木 直史 (正会員)

1959年生. 1983年京都大学大学院工学研究科修士課程情報工学専攻修了. 1984年同大工学部助手. 1990～91年スタンフォード大学客員研究員. 1991年京都大学工学部助教授. 1994年名古屋大学工学部助教授, 現在に至る. 工学博士. 論理回路, ハードウェアアルゴリズム, 算術演算回路などの研究に従事. 本会坂井記念特別賞, 日本IBM科学賞, 本会論文賞, 電子情報通信学会論文賞などを受賞. 電子情報通信学会, IEEE各会員.